# Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems

Gérald Oster, Pascal Molli, Pascal Urso, Abdessamad Imine

▶ **To cite this version:**

Gérald Oster, Pascal Molli, Pascal Urso, Abdessamad Imine. Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems. IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006, Nov 2006, Atlanta, Georgia, USA, pp.1-10, 10.1109/COLCOM.2006.361867 . inria-00109039

**HAL Id: inria-00109039**

**https://inria.hal.science/inria-00109039**

Submitted on 7 Nov 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems

Gérald Oster
Institute for Information Systems
ETH Zurich
Email: osterg@inf.ethz.ch

Pascal Molli, Pascal Urso and Abdessamad Imine
Nancy-Université
LORIA
Email: {molli,urso,imine}@loria.fr

***Abstract*— In collaborative editing, consistency maintenance of the copies of shared data is a critical issue. In the last decade, Operational Transformation (OT) approach revealed as a suitable mechanism for maintaining consistency. Unfortunately, none of the published propositions relying on this approach are able to satisfy the mandatory correctness properties $TP_1$ and $TP_2$ defined in the Ressel's framework. This paper addresses this correctness issue by proposing a new way to model shared state by retaining tombstones when elements are removed. An instantiation of the proposed model for a linear data structure and the related transformation functions are provided.**

## I. Introduction

Collaborative editing systems allow users to edit the same document from multiple sites across Internet. Depending on the work context, users can work synchronously or asynchronously. Synchronous collaboration is also called real-time editing since when a user performs some modifications on the document, these modifications are instantly sent to other users who can see them without any delay. In the contrary, in asynchronous collaboration, users may not work at the same time. They work in isolation: they can decide when to publish their modifications and when to integrate modifications performed by other users.

In these systems, the shared documents are commonly replicated at multiple sites. Parallel modifications on these copies may happen and therefore potential inconsistencies may occur. One of the main issues in collaborative editing is how to maintain consistency of shared documents copies. Consistency maintenance mechanisms are classified in two categories depending on whether they are pessimistic or optimistic.

Pessimistic approaches try to give the impression there is only one highly available copy in the whole system. Only one copy – or part of a copy – can be edited at the same time while all the copies can be read. This principle is generally realised using a locking mechanism such as in database transaction systems [1] or in turn-taking protocols [7]. Over the years, this class of mechanisms revealed unsuitable for collaborative editing even though they ensure strong consistency. They set too many restrictions on collaborative interactions – no concurrent updates are allowed – and they do not support work disconnected from the network. Furthermore, the large delays between requesting and acquiring a lock make these mechanisms not appropriate for real-time editing.

On the contrary, optimistic approaches [24] are more suitable for collaborative editing since they tolerate divergence between copies and ensure that the copies converge at a later time. In particular, the so-called approach Operational Transformation [5] (OT) has been specifically designed to fulfil the requirements of collaborative editing. In this approach, updates performed by one user are applied on the local copy without any delay. Next, they are broadcast to other copies whether synchronously or asynchronously. Finally, updates have to be executed on the remote copies. Incoming updates are transformed according to concurrent updates that might have been performed in the meantime on these remote copies. These transformations are computed in a way that will ensure convergence of the copies. It is worth to point out that the local response time is not sensitive to network latencies since local updates are executed immediately.

Since the initial work of Ellis et al. [5], several OT frameworks [23], [29], [13], [15] have been proposed. The first OT framework was developed in 1996 by Ressel et al. [23]. This framework makes a strong separation between a generic integration algorithm and specific transformation functions. Transformation functions depend on the type of shared data, whereas the integration algorithm does not. If a developer of a collaborative editor wants to provide sharing of one specific data type, he has to write the adequate transformation functions and prove they conform to the two correctness properties $TP_1$ and $TP_2$. Under these conditions, the integration algorithm will ensure that causality between operations is preserved and convergence of copies is achieved.

Unfortunately, satisfying $TP_2$ is very difficult. In [10], [13], it has been proven that all proposed transformation functions do not satisfy this property.

A proposed solution to this problem was to require only $TP_1$ property and fix a total ordering on the integration of operations. The SOCT4 algorithm [30] implements this strategy while conserving the generality of the OT approach. The SO6 synchronizer [25] based on SOCT4 demonstrated the use of this OT's strength to build a tool very similar to CVS capable to reconciliate a file system containing text files and XML documents. Unfortunately, building the total ordering requires a central site or a stable pool of sites [4]. These new constraints prevent SOCT4 to be used in a pure decentralized environment such as a peer-to-peer network.

In order to escape from $TP_2$ property, other frameworks have been proposed [16], [8], [20]. Although these frameworks are very interesting, their integration algorithms are closely bound to linear-structure properties of shared document for which they have been designed. Thus, in order to handle new types of data or to allow more operations to be performed on the data, the integration algorithm has to be modified and some new correctness properties must be determined and proven on it. Hence, if they propose new ways to model and solve consistency problems, they do not have yet the generality of the original Ressel's framework.

This paper presents the first set of transformation functions that ensures $TP_1$ and $TP_2$. This new set of transformation functions resulted from the integration of the tombstone approach [20] with the Ressel's framework. Therefore, the new set of transformation functions, called TTF (Tombstones Transformation Functions), demonstrates that the Ressel's framework can be instanciated.

This paper is structured as follows. We start by presenting the Ressel's framework. We then describe our approach and discuss its correctness. After, we relate our approach to previous works. Finally, we conclude and point out some future work.

## II. Background and Open Problems

A collaborative editing system consists of a set of participant systems connected by a communication network. In the following, a participant system is called a site. There is one site per user. Usually a site correspond to one user's workstation. However, sometimes one computer might run multiple sites. Because collaborative editing systems require high responsiveness and should offer support for users to work in isolation, the shared data are replicated at every site. In other words, a collaborative editing system is modelled as follows. It considers $n$ sites, each site owns a copy of shared data. When a site performs an update, it generates a corresponding operation. Every operation is processed in four steps: (i) execution on one site, (ii) broadcast to other sites, (iii) reception by other sites, (iv) execution on other sites.

The OT approach distinguishes two main components:

- an *integration algorithm*. This algorithm is in charge of reception, diffusion and execution of operations. When necessary, it calls transformation functions. This algorithm does not depend on type of replicated data ;
- a set of *transformation functions*. These functions merge concurrent modifications in transforming two concurrent operations in order to execute them in a serial order. These functions are specific to a particular type of replicated data such as string of characters, XML document or file system.

The study of this paper is restricted to shared document relying on a linear structure. Problems and proposed solutions can be generalised to more complex structures such as hierarchical structures as shown in [3], [18], [9]. Without loss of generality, in the remaining of this paper, the shared document is considered to be a string of characters. A string of characters might be updated by performing two kinds of operations:

- $ins(p, c)$ inserts a new character $c$ at position $p$ in the string.
- $del(p)$ removes the character located at position $p$ in the string.

The first character of a string is assumed to be located at position 1.

Over the years, consistency maintenance in OT has been refined to the guarantee of two criteria: *causality preservation* and *convergence of copies*.

Considering two operations $op_1$ and $op_2$, operation $op_1$ is said to *precede* $op_2$ if and only if $op_2$ is generated on a copy after $op_1$ was executed on this copy. Subsequently, $op_2$ may depend on effects of execution of $op_1$. *Causality preservation* criterion ensures that all operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [12], will be executed in the same order on every copy. Two operations $op_1$ and $op_2$ that are not related by a precedence relation (neither $op_1$ precedes $op_2$, nor $op_2$ precedes $op_1$) are said to be concurrent.

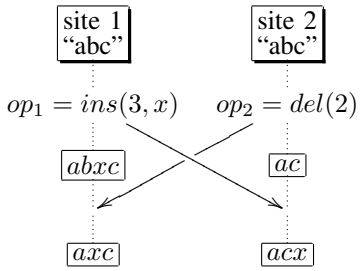


Fig. 1. Divergence problem.

Two concurrent operations can be executed in different order on two different copies. Consequently, when an operation is received on one site, the current state of shared object may be different from the one where the operation has been generated. Thus, executing this operation in its generated form on a remote site may not preserve its effects and the copies may not converge. Figure 1 illustrates such a scenario. Operations $op_1$ and $op_2$ have been generated concurrently on two different copies of the string "abc". $op_1$ inserts an $x$ at position 3 to obtain the string "abxc", while $op_2$ removes the character $b$ located at position 2. If these operations are executed in their original form when they are received by other sites, two divergent states "axc" and "acx" are obtained at site 1 and site 2 respectively, as depicted in Figure 1.

In order to solve this kind of consistency problems, Ellis et al. [5] introduced a transformation function $T$. This function is used to transform remote operations when they arrive on a site. Remote operations are transformed regarding concurrent operations that were already executed on local copy.

For instance, in our previous example, $op_1$ is not any more executed in its generation form when it arrives on site 2, but it is transformed regarding concurrent operations, in our case

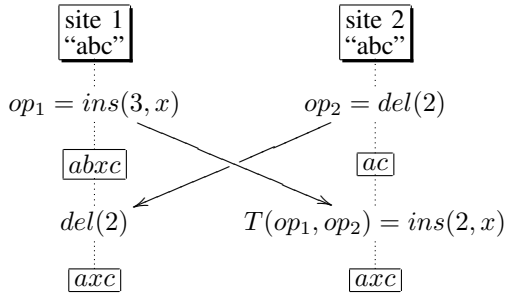


Fig. 2. Convergence by transforming.

operation $op_2$. As $op_2$ removed a character located before the insertion position of $op_1$, the insertion position of $op_1$ is decreased of one position to take into account previous execution of $op_2$. Consequently, on site 2, operation $op'_1 = ins(2, x)$ has to be executed (see Figure 2). Intuitively, the transformation used is defined as follows:

$T(ins(p_1, c_1), del(p_2)) :-$
  **if** $(p_1 \leq p_2)$ **return** $ins(p_1, c_1)$
  **else return** $ins(p_1 - 1, c_1)$

**Definition 1** *A transformation function*[1] *T takes two concurrent operations as parameters. These two operations, namely $op_1$ and $op_2$, must be defined on a same state $S$. The function computes a new operation $T(op_1, op_2)$ equivalent to $op_1$ – i.e. has the same effects – but defined on the state $S'$ resulted from the execution of $op_2$ on state $S$.*

Later, Ressel et al. [23] identified two properties $TP_1$ and $TP_2$ which must be satisfied by transformation functions for ensuring convergence of the copies independently of the integration order of concurrent operations.

**Definition 2** *For every pair of concurrent operations $op_1$ and $op_2$ defined on the same state, the transformation function $T$ satisfies $TP_1$ property if and only if :*

$$op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$$

*where $op_i \circ op_j$ denotes the sequence of operations containing $op_i$ followed by $op_j$ ; and where $\equiv$ denotes equivalence of the two sequences of operations.*

This first property $TP_1$ expresses equivalence between two sequences. Given two concurrent operations $op_1$ and $op_2$, the execution of the sequence of $op_1$ followed by $T(op_2, op_1)$ on a state $S$ must produce the same state as the execution of the sequence of $op_2$ followed by $T(op_1, op_2)$.

**Definition 3** *For every three concurrent operations $op_1$, $op_2$ and $op_3$ defined on the same state, the transformation function $T$ satisfies $TP_2$ property if and only if:*

$$T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$$

[1]In the literature, this function is also called forward transformation or inclusion transformation.

This second property $TP_2$ stipulates *equality* between two operations transformed with regard to two equivalent[2] sequences of operations. Given three operations $op_1$, $op_2$ and $op_3$, the transformation of $op_3$ with regard to the sequence formed by $op_2$ followed by $T(op_1, op_2)$ must give the same operation as the transformation of $op_3$ with regard to the sequence formed by $op_1$ followed by $T(op_2, op_1)$.

Ressel et al. [23] demonstrated that these two properties $TP_1$ and $TP_2$ are sufficient to ensure convergence of copies independently of the order in which concurrent operations are transformed.

With a correct set of transformation functions, the integration algorithm ensures consistency and the resulting collaborative editing tools would be reliable. Indeed, most of the OT integration algorithms have been proven [26], [17] to ensure convergence of copies if the underlying transformation functions satisfy the properties. Unfortunately, currently none of the transformation functions proposed are correct regarding $TP_1$ and $TP_2$ properties. Imine et al. [10] used a formal approach based on a theorem prover to check correctness of all previously published set of transformation functions. Counter-examples for each set of transformation functions have been provided in [10]. Li et al. [13] gave also a counter-example for the transformation functions proposed by Imine et al. [10]. In Section IV, we will give a counter-example for [13]. Therefore, currently there is no correct transformation functions and consequently the operational transformation cannot be used to build a safe decentralized collaborative editing system.

In this paper, we present the first set of transformation functions satisfying both $TP_1$ and $TP_2$ properties. Consequently, these transformation functions could be used with any integration algorithm previously published.

## III. Transformation Functions

All the transformation functions previously published do not satisfy $TP_2$ property because they fail to solve a variant of the same problem. Even if some counterexamples are more complicated – involving partial concurrency between operations – the basic problem remains the same. This problem, presented in Figure 3, involves three concurrent operations : $ins(2, x)$, $del(2)$ and $ins(3, y)$.

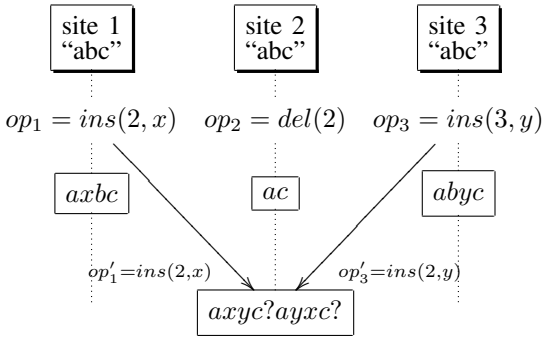


Fig. 3. Common problem.

[2]These two sequences are equivalent by transformation.

When $op_1 = ins(2, x)$ is received on site 2, it is transformed according to $op_2$. This transformation does not change insertion position of $op_1$, and, returns as result the operation $op'_1 = ins(2, x)$. When $op_3 = ins(3, y)$ is received on site 2, it is transformed according to $op_2$. Its insertion position is decreased and thus it becomes $op'_3 = ins(2, y)$. At this point on site 2, if the algorithm has to transform $op'_1$ according to $op'_3$, or $op'_3$ according to $op'_1$, then it must break a tie because their current positions of insertion are equal. To break this tie, all the approaches choose a different way: initial positions for IMOR [10], sets of concurrent deletions for Suleiman et al. [26] or state difference for Li et al. [13]. Unfortunately, all these approaches fail to order correctly $x$ and $y$ in some cases. And, all the counter-examples [10], [13] violating $TP_2$ property are only instances of this tie. Nevertheless, on the initial state, character $b$ obviously separates $x$ and $y$; $b$ is called landmark character by Li et al. [15]. Consequently, the algorithm must insert $x$ in front of $y$ on all sites.

### A. (U)ncompacted-TTF Model

Our idea is to keep the deleted character $b$ as a tombstone. If a character is deleted, the system maintains useful information about its former position but not its whole content. Tombstones are well known in distributed systems. For example, they are heavily used in Usenet[3] to make conflicts between update/delete operations non ambiguous [24].

For a character string, it is equivalent to keep the character in its position and mark the character as invisible. If the shared data is a more complex linear data structures such as a list of lines instead of list of characters, it means that the system will maintain the identity of the line but not its content.

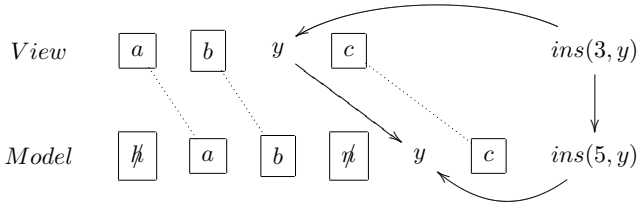



Fig. 4. Relation between view and model in TTF.

Consequently, hidden characters must remain present in the model of the string from a site, but are hidden from the view of the string seen by the user. The model of the string $S$ becomes a sequence of ordered pairs $(character, visible)$ where $visible$ is a boolean attribute. This model and its behaviour are illustrated in Figure 4. Suppose a user inserts the character $y$ between $b$ and $c$, then the operation $ins(3, y)$ is generated on the view, but this operation is executed on the model as $ins(5, y)$ since the characters $h$ and $n$ previously existed. Operation $ins(5, y)$ is broadcast and transformed on all other sites.

To perform the conversion between the position of an operation in the model and the position in the view, the system uses the following function:

[3]Usenet is the system network in charge of replicating newsgroups.

```
viewToModel(int pview, S) : int {
  int n=1, j=1;
  while ((j ≤ length(S)) and (n<pview or not S[j].visible)) {
    if (S[j ]. visible ) n++;
    j++;
  }
  return j ;
}
```

Suppose the system has to execute a newly operation $ins(p, c)$ generated from the view. Then, on the model, it has to find out the first position located after traversing $p$ visible characters and then skips all removed characters (tombstones) situated after this current position. The complete functions to execute local operations and remote operations are described in Figure 5 and in Figure 6. The function shiftRight(S,p) makes a new room in the string $S$ at position $p$ by shifting to right every character in the range $[p; length(S)]$. In the same way, shiftLeft(S,p), used later in this paper, removes the element located at position $p - 1$ in the string $S$ by shifting to left every character in the range $[p; length(S)]$. Note that these functions are commonly required to insert or remove an element in string, even without using the TTF data model.

```
executeLocal(ins(x,pview), S) {
   int pmodel=viewToModel(pview);
   shiftRight(S,pmodel); // make room for the new character
   S[pmodel]=x;
   broadcast(ins(x,pmodel));
}
executeLocal(del(pview), S) {
   int pmodel=viewToModel(pview);
   S[pmodel].visible=false;
   broadcast(del(pmodel));
}
```

Fig. 5. Local executions in the U-TTF model.

Executing a received operation is different from executing a local operation. Indeed, position parameter of a remote operation is located in the model while position of a local operation is associated to the view. So we define two functions executeRemote(ins(x,p), S) and executeRemote(del(p), S) described in Figure 6.

```
executeRemote(ins(x,pmodel), S) {
   shiftRight(S,pmodel); // make room for the new character
   S[pmodel]=x;
}
executeRemote(del(pmodel), S) {
   S[pmodel].visible=false;
}
```

Fig. 6. Remote executions in the U-TTF model.

This strategy leads to the trivial transformation functions presented in Figure 7. An additional parameter $sid_i$ has been added to operations. It identifies in a unique manner the site on which the operation has been generated. Identifiers of two

insert operations are compared in order to break the tie when two insertions have been performed concurrently at the same position.

$T(ins(p_1, c_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  **if** $(p_1 < p_2)$ **return** $ins(p_1, c_1, sid_1)$
  **else if** $(p_1 = p_2$ **and** $sid_1 < sid_2)$ **return** $ins(p_1, c_1, sid_1)$
  **else return** $ins(p_1 + 1, c_1, sid_1)$

$T(del(p_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  **if** $(p_1 < p_2)$ **return** $del(p_1, sid_1)$
  **else return** $del(p_1 + 1, sid_1)$

$T(ins(p_1, c_1, sid_1), del(p_2, sid_2)) :-$
  **return** $ins(p_1, c_1, sid_1)$

$T(del(p_1, sid_1), del(p_2, sid_2)) :-$
  **return** $del(p_1, sid_1)$

Fig. 7. TTF transformation functions.

Since the execution of a $del()$ operation replaces the removed character with a tombstone, it does not affect the position of other characters in the string. Consequently, performing a transformation of an operation $op$ according to a $del()$ operation does not modify the effect position of $op$.

Transformation functions of any operation according to an $ins()$ operation are defined as originally done by Ressel et al. [23]. When two insert operations have the same position $p$, the character produced by the site with the lower range is inserted at $p$; the other character is inserted at position $p + 1$.

We have proven that TTF ensure $TP_1$ and $TP_2$. Due to space limitations, the proof is not included in this paper. This proof was built using the automatic theorem prover SPIKE [2], [11]. The full specification and how the proof is built by the theorem prover is available in [19]. But even without SPIKE, it is easy to see that, in opposite to traditional transformation functions, the TTF are monotonic transformations of the effect position of operations since they only compute additions. Hence, the position of one character will grow monotonically to the same value independently of the equivalent transformation path taken.

This monotonic property has another interesting consequence: TTF preserve order relationships between characters which is considered in [13] as an instantiation of the intention preservation criterion defined by Sun et al [29]. In [13], the intention of $ins(p, c)$ operation is expressed by the relation $\prec$. If one user generates $op = ins(p, c)$ on a site where $x$ is visible at a position less than $p$, and $y$ is visible at a position greater or equal to $p$, then the ordering $x \prec c \prec y$ is set. Concerning a $del(p)$ operation executed on a string $S$, its intention is to remove the character $S[p]$. Please note that in our model $del(p)$ does not remove the character but makes it invisible. Preserving intention of $ins()$ operations means that the $\prec$ relations hold on all further states. Since in our model $del()$ operations do not affect the positions of characters, the $\prec$ relations will always be preserved on the generation site of $op$. Moreover, using our TTF all copies on all sites will eventually converge to the same string. Consequently, for any operation $ins(p, c)$ the ordering $x \prec c \prec y$ will be preserved in any further states. In other words, TTF can ensure intention preservation criterion as defined in [13].

### B. Inverses of Transformation Functions

Some algorithms such as SOCT2 [26], GOT [29] and GOTO [28] required to define additional functions called "exclusion transformation" [29]. These functions are the inverses of the transformation functions. Since the TTF transformation functions are injective functions, defining their inverse functions is straightforward. Figure 8 gives the definitions of the inverses of the TTF transformation functions.

$T^{-1}(ins(p_1, c_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  **if** $(p_1 < p_2)$ **return** $ins(p_1, c_1, sid_1)$
  **else if** $(p_1 = p_2$ **and** $sid_1 < sid_2)$ **return** $ins(p_1, c_1, sid_1)$
  **else return** $ins(p_1 - 1, c_1, sid_1)$

$T^{-1}(del(p_1, sid_1), ins(p_2, c_2, sid_2)) :-$
  **if** $(p_1 < p_2)$ **return** $del(p_1, sid_1)$
  **else return** $del(p_1 - 1, sid_1)$

$T^{-1}(ins(p_1, c_1, sid_1), del(p_2, sid_2)) :-$
  **return** $ins(p_1, c_1, sid_1)$

$T^{-1}(del(p_1, sid_1), del(p_2, sid_2)) :-$
  **return** $del(p_1, sid_1)$

Fig. 8. Inverses of TTF transformation functions.

The preconditions of the inverse of a transformation function required that the first parameter operation resulted from a previous transformation according to the second one. Indeed, it is not allowed to call this inverse function in order to swap two operations causally dependent because these operations were not concurrent, and thus were not previously "serialised" using transformation functions. For instance, consider the string of characters "abc" and two operations $op_1 = ins(2, x)$ and $op_2 = del(2)$ executed in this order on the string of characters. The resulting state after the executions of $op_1$ and $op_2$ is "abc". Since, these two operations are causally dependent as $op_2$ deletes the character inserted by $op_1$, it is forbidden to try to swap these two operations in order to execute them in the reverse order ($op_2$ then $op_1$). Performing such transformations will compute the sequence of operations $[del(1), ins(1, x)]$ whose execution will lead to the wrong state "xbc". These preconditions are always ensured by OT integration algorithms such as SOCT2 for example.

### C. Optimising TTF Approach

Compared to existing approach, the function viewToModel() has to be computed each time a character is generated. Its time complexity is linear to the size of the model. In case of asynchronous editing this complexity has no impact since operations are generated when user edition is finished.

In case of real time editing, this extra time will slow down the local response time if the text is long. However, we can obtain a good response time by a simple optimisation. In an editor, local operations are relative to the caret position. Indeed, the viewToModel() computes the position of the caret in the model. This position could be stored as the global variable caretPosition. And, its value is updated when the user moves the caret in the text with the procedure updateCaret(oldPos, newPos, S). The parameters oldPos and newPos are the old and new caret positions in the view presented to the user. Therefore, before the procedure updateCaret() is called, the global variable caretPosition is equal to the position in the model corresponding to the position oldPos in the view. And, after updateCaret() was executed, caretPosition is equal to the position in the model associated to newPos in the view.

```
updateCaret(int oldPos, int newPos, S) {
  int i = oldPos;
  if (newPos ≥ oldPos) { // move right
    while (i <> newPos) {
      if (S[caretPosition]. visible ) i++;
      caretPosition++;
    }
    // skip removed char
    while (caretPosition ≤ length(S)
           and not S[caretPosition]. visible )
      caretPosition++;
  } else { // move left
    while (i <> newPos) {
      if (S[caretPosition−1]. visible ) i−−;
      caretPosition−−;
    }
  }
}
```

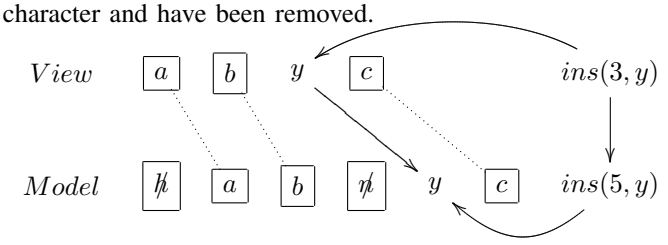
The executeLocal() functions could be rewritten as follows:

```
executeLocal(ins(x,caretPosition), S) {
  shiftRight(S,caretPosition);
  S[caretPosition] = x;
  broadcast(ins(x,caretPosition));
}
executeLocal(del(caretPosition), S) {
  S[caretPosition]. visible = false;
  broadcast(del(caretPosition));
}
```

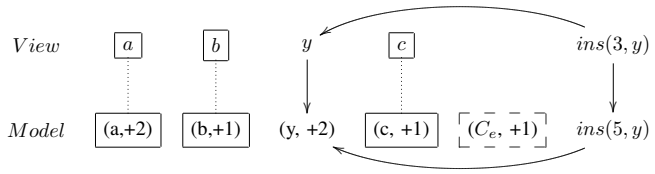
### D. (D)elta-TTF Model

Although tombstones is a very easy solution for the OT approach, TTF retains a tombstone to mark a deleted character. So, the space overhead of tombstones grows indefinitely. Two solutions are generally used to solve this issue in distributed systems. The first one is based on an expiration period which is associated to each tombstone. A tombstone is definitively suppressed as soon as its period expires. It is well-known that this method is unsafe. An operation assuming presence of some tombstones might arrive after their expiration periods, in this case this operation cannot be executed properly. The second solution employs a two-phase protocol to purge safely the tombstones as described in [24]. Unfortunately, such a protocol requires all sites must be alive for the algorithm to make progress. In other words, it means that all participants using the collaborative editing system must be and stay connected until the garbaging is finished. This assumption is not suitable for asynchronous collaborative editing systems.

In our context, this problem is equivalent to managing a sparse array where deleted characters are considered as zero entries. The basic idea when storing this kind of array is to only store the non-zero entries as opposed to storing all entries. Hence, we present now another model to store the local string. Each visible character keeps an integer value equals to 1 + the number of invisible characters located between it and the visible character preceding it. This new model is depicted in Figure 9(b). This new model requires to add a special invisible character named $C_e$ which is used to keep the number of characters plus one which were located after the last visible character and have been removed.




(a) TTF uncompacted model




(b) TTF delta model

Fig. 9. Difference between U-TTF and D-TTF models.

The viewToModelD function is rewritten as follows:

```
viewToModelD(int pos, S) : int {
  return ∑_{k=1}^{pos} S[k].offset;
}
```

The functions of Figure 10 describe the execution of a local insert or delete operation. $S$ is a sequence of pair $(character, relativeposition)$. S[p].offset returns the relative position of the character stored at position $p$ in $S$. Figure 11 defines how to execute a received operation.

Theoretical complexity of the execution of the viewToModelD function is linear. However, we can apply the same optimization we used for the uncompacted model. The position of the caret in the model is stored and updated when user moves his caret in his text editor.

## IV. Related Work

During the last decades, the operational transformation model has triggered a growing enthusiasm for maintaining

```
executeLocal(ins(x,p), S) {
  shiftRight(S, p);  // make room for the new character
  S[p] = (x, S[p+1].offset);
  S[p+1].offset = 1;
  broadcast(ins(x, viewToModelID(p, S)));
}
executeLocal(del(p), S) {
  S[p+1].offset = S[p+1].offset + S[p].offset;
  int pos = viewToModelID(p, S);
  shiftLeft(S, p+1); // remove the character entry
  broadcast(del(pos));
}
```

Fig. 10. Local executions in the D-TTF model.

```
executeRemote(ins(x,p), S) {
  int sum = 0, i = 0;
  while (i ≤ length(S) and sum < p) {
    i++;
    sum += S[i].offset;
  }
  shiftRight(S, i);
  S[i] = (x, p−(sum−S[i+1].offset));
  S[i+1].offset = sum−p+1;
}
executeRemote(del(p), S) {
  int sum = 0, i = 0;
  while (i ≤ length(S) and sum < p) {
    i++;
    sum += S[i].offset;
  }
  if (sum==p) {
    S[i+1].offset = S[i+1].offset + S[i].offset;
    shiftLeft(S,i+1);
  }
}
```

Fig. 11. Remote executions in the D-TTF model.

consistency in collaborative editing systems. Since the initial work of Ellis et al. on dOPT [5] algorithm was found false regarding the $TP_2$ properties, two ways of research are explored.

The idea of the first approach is to avoid the need of $TP_2$ property. There are mainly two works based on this approach: the GOT [29] algorithm and the SOCT4 [30] algorithm. In GOT, all operations will be eventually executed in the same total order on every site. In this manner, convergence of copies is ensured even if transformation functions do not satisfy $TP_1$ nor $TP_2$ property. These two properties are not required because on every site each operation will be transformed according to concurrent operations in the same order. This algorithm has one drawback. Since in OT approach local operations are always executed immediately on the local copy, some local operations could be executed before the arrival of some remote operations. However, the executions of these remote operations might precede in total order the execution of the local operations. To solve this issue, GOT uses an undo-do-redo mechanism [29] for undoing the local operations, then executes the remote operations and finally re-executes the undone operations. This undo-do-redo mechanism requires the exclusion transformation functions which of course must satisfy the reversibility property. Unfortunately, until now there are no transformation functions satisfying this property. The SOCT4 algorithm relies also on a total order, but in place of the undo-do-redo mechanism, it restricts the broadcast of local operations. Indeed, in SOCT4, a site can send its local operations only if it has integrated all those remote operations that precede in total order the local operations. This algorithm requires only the $TP_1$ property to be satisfied on transformation functions to ensure convergence. Unfortunately, the continuous total order required by SOCT4 is generally implemented using a central time-stamper which restricts considerably the collaboration interaction practices. Moreover, if there exist transformation functions satisfying $TP_1$ condition, they do not achieve intention preservation as defined in [13]. Hence, all copies will eventually converge to a unique state that might violate this definition of intentions of operations. Since the TTF transformation functions preserve intentions as defined in [13] and are reversible, they could be used to fix correctness issues of current collaborative systems based on GOT or SOCT4 algorithms.

The purpose of the second approach is to find transformation functions satisfying $TP_2$ property. A lot of integration algorithms, such as adOPTed [23], SOCT2 [26] or GOTO [28], were proposed assuming that transformation functions will satisfy the $TP_2$ property. Each algorithm was proposed in association with some transformation functions. None of these transformation functions are correct regarding $TP_2$ properties as summarised in [10]. Authors of [10] have proposed some transformation functions, but Li et al [13] have found a counter-example for them violating $TP_2$ and intention preservation.

A $TP_2$ counter-example was found for each existing transformation functions except SDT transformation functions. SDT [13] stands for State-Difference Transformations. Authors wrote that SDT ensures $TP_1$ and $TP_2$. Unfortunately, we found a counter-example with the help of the SPIKE theorem prover [19].

When the system has to transform one operation $op_1$ according to a second concurrent operation $op_2$, it performs the following steps. First, it identifies the last common state (LSP) from which both concurrent operations were executed. Then, it computes the sequence $SD$ of operations that could be executed on LSP to lead to the state on which $op_2$ is defined. Next, it computes $\beta(op_1)$ the potential effect position of $op_1$ on LSP in excluding $SD$ from $op_1$. It performs the same with $op_2$ to get $\beta(op_2)$. Finally, these two positions are compared in the transformation for inferring the shift of the effect position of $op_1$.

The Figure 12 illustrates the SDT counter-example. The assumption $\beta(op_{41}) = \beta(op_2) = \beta(op_3)$ could be satisfied if $op_3$ is generated on a state preceding the definition state of $op_2$ and $op_3$. Since $\beta(op_3) = \beta(op_{41})$, SDT functions compared the effect positions to break the tie between the two concurrent operations $op_3$ and $op_{41}$. On site 1, the positions

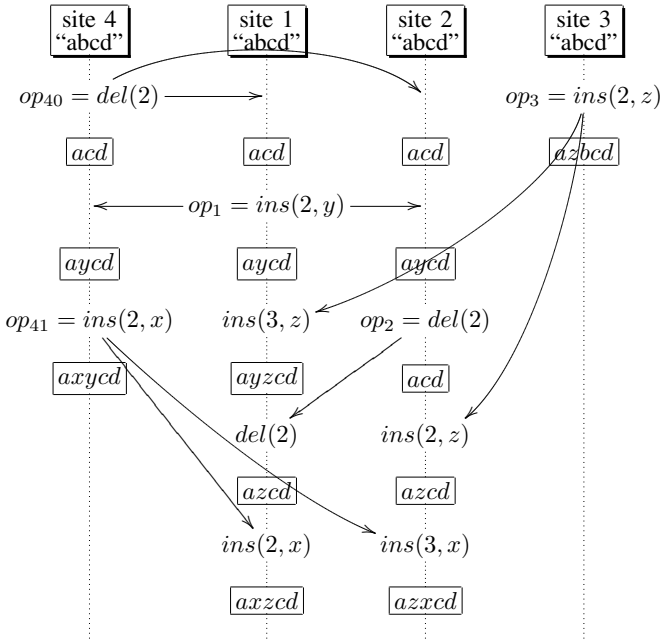



Fig. 12. SDT counter-example

satisfy the relation $p(op_{41}) < p(op_3)$. Therefore, positions determine the result of the transformations. But, on site 2, the positions are equal, i.e. $p(op_{41}) = p(op_3)$, and then the site priority is used to break the tie. Since on two sites two different mechanisms are used to break the tie, a tricky choice in site priorities will lead to the divergence illustrated in Figure 12. We also verified the proof published [14]. We found that the proof of Lemma 5 is incomplete. Indeed, in this proof, authors consider two cases: $\overline{O_3} \rightarrow (O_1 \parallel O_2)$ or $\overline{O_3} \parallel (O_1 \parallel O_2)$. Unfortunately, the proof of Lemma 5 does not consider the case where $\overline{O_3} \rightarrow O_1$ and $(\overline{O_3} \parallel O_2)$ and $(O_1 \parallel O_2)$. The counter-example presented in Figure 12 instantiates this latter case with: $O_3 = op_2 = del(2)$, so $\overline{O_3} = op_1 = ins(2, y)$ ; $O_2 = op_3 = ins(2, z)$ ; $O_1 = op_{41} = ins(2, x)$.

Consequently, it means that, currently, only TTF can be used to instantiate the Ressel's framework. Other research proposals consider that the solution is not to instantiate the Ressel's framework but to propose a new framework.

ABT [16] is an alternative approach to address consistency problems of existing OT algorithms. The first motivation of this work was the difficulty to find transformation functions that verify $TP_1$ and $TP_2$. In this paper, we have shown that $TP_1$ and $TP_2$ can be easily achieved by using simple transformation functions. The second motivation of this work was that intentions of operations must be preserved during transformations, but they were not defined in the Ressel's model and were not formalized when defined in [29]. Anyway, we proved in Section III-A that TTF preserve order relationships between characters. These relationships correspond to the effect relations as defined in ABT. Hence, we presented a set of transformation functions that verify all requirements described in [16] while keeping the Ressel's model and related integration algorithms. However, the ABT model is defined only for linear structures while the Ressel's model is independent of structures of shared data.

There is another approach which does not transform operations: the Mark & Retrace technique [8]. When a remote operation has to be integrated, the document's address space is retraced to the state at the time the operation was generated. This *retracing* is done by marking effective the characters which were present at the generation time – even the ones which have been deleted afterwards –, and ineffective the characters which are inserted concurrently. Then, in the case of an insertion, the new character has to be inserted between its previous and next character. But, since some concurrent inserted and deleted characters might be present between its previous and next character, a *range-scan* function is applied to order totally these characters. These functions and their proofs of correctness clearly depend on the linear structure of the shared object. Thus, applying this technique to other types of structure will require designing new algorithms and proving them. Moreover, there is no easy optimisation to perform in order to garbage operations or deleted characters. Furthermore, ensuring convergence for specific shared data structures might require cancelation or transformation of operations. In these cases, serialisation mechanism are not sufficient. For instance, consider the case of a file system as a shared data. Suppose two copies of this file system are concurrently modified: one operation adds a new file $b$ while another operation creates a new directory also named $b$. Using the Mark & Retrace technique to find a serial execution order of these two operations is not possible since the execution of one operation invalidates the precondition of the other operation – the precondition of both operations requires that the name $b$ is not used –. Therefore, to make the copies converge, one of these two operations has to be canceled, and so, one of the concurrent updates is lost. By using operational transformations as in our approach, a solution could be provided that combines both updates. Such a combination could lead to a state where the file is named $b$ and the directory is renamed $b\#$. If users are not satisfied with this solution, they can further refine it since no work was last.

The WOOT framework [20] works in a quite similar way to the *range-scan* function of Mark & Retrace, except that it does not require the execution of the retracing process. The previous and the next characters between which an operation is performed are found in the string using their unique identifiers. This feature eliminates the requirement of state vectors since they might be a weak point in large-scale systems. However, the garbage collection of operations and characters marked for deletion are still open issues. And, this framework is not generic since it relies strongly on some structural precedence relation on characters.

## V. Conclusions and Future Work

In this paper, we proposed the Tombstone Transformation Functions for maintaining consistency in collaborative

editing systems based on the operational transformation approach. These functions satisfy the correctness properties $TP_1$ and $TP_2$, and consequently they are able to preserve intentions while ensuring copies convergence. These functions can be used with existing integration algorithms such as adOPTed [23], SOCT2 [26] or SOCT4 [30] and therefore could be easily used to replace the wrong transformations used in existing systems.

In this paper we presented our solution for linear structures, but the simplicity of the mechanism offers support for the extension to other data structures.

In collaborative editing systems, the ability to undo operations [21] is a widely used feature. In the operational transformation community, several propositions [23], [22], [27], [6] have already been made. It is worth to point out that maintaining tombstones greatly simplifies the design of an undo mechanism. Indeed, since all the locations of deleted elements are kept, it is easy to restore them at their right place. One of the main differences with the previous approaches is that undoing the deletion of an element is different from inserting this element. This means a new operation has to be added in the model. We are currently working on the verification of the correctness properties for undo mechanism defined in [6] for our approach [31].

As future work we also plan to extend our TTF to string-wise operations as described in [29] and hierarchical data structures as presented in [3], [9].

Our aim is to deploy a collaborative editing system on a peer-to-peer network. Actually, most of the OT algorithms that do not require a central site are based on version vectors to times updates. Unfortunately, version vectors do not scale and cannot be used in a system with a large number of sites. We are investigating ways of extending operational transformation algorithms to fit the requirements of this kind of network.

## ACKNOWLEDGMENTS

We wish to thank Claudia-Lavinia Ignat for her very valuable comments and suggestions which helped us to improve the presentation of this article.

## REFERENCES

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Boston, MA, USA, 1987.

[2] A. Bouhoula. Automated Theorem Proving by Test Set Induction. *Journal of Symbolic Computation*, 23(1):47–77, January 1997.

[3] A. H. Davis, C. Sun, and J. Lu. Generalizing Operational Transformation to the Standard General Markup Language. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2002*, pages 58–67, New Orleans, Louisiana, USA, November 2002. ACM Press.

[4] X. Défago, A. Schiper, and P. Urbán. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys*, 36(4):372–421, December 2004.

[5] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. *SIGMOD Record: Proceedings of the ACM SIGMOD Conference on the Management of Data - SIGMOD'89*, 18:399–407, May 1989.

[6] J. Ferrié, N. Vidot, and M. Cart. Concurrent Undo Operations in Collaborative Environments Using Operational Transformation. In *Proceedings on the Conference on Cooperative Information Systems - CoopIS 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 155–173, Agia Napa, Cyprus, October 2004. Springer Verlag.

[7] S. Greenberg. Personalizable Groupware: Accommodating Individual Roles and Group Differences. In *Proceedings of the European Conference of Computer-Supported Cooperative Work - ECSCW'91*, pages 17–32, Amsterdam, Netherlands, September 1991. Kluwer Academic Publishers.

[8] N. Gu, J. Yang, and Q. Zhang. Consistency Maintenance Based on the Mark & Retrace Technique in Groupware Systems. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2005*, pages 264–273, Sanibel Island, Florida, USA, November 2005. ACM Press.

[9] C.-L. Ignat and M. C. Norrie. Customizable Collaborative Editor Relying on treeOPT Algorithm. In *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW 2003*, pages 315–334, Helsinki, Finland, September 2003. Kluwer Academic Publishers.

[10] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving Correctness of Transformation Functions in Real-Time Groupware. In *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW 2003*, pages 277–293, Helsinki, Finland, September 2003. Kluwer Academic Publishers.

[11] A. Imine, P. Molli, G. Oster, and P. Urso. VOTE: Group Editors Analyzing Tool. In *Proceedings of the International Workshop on First-Order Theorem Proving - FTP 2003*, volume 86, Valencia, Spain, June 2003. Elsevier B.V.

[12] L. Lamport. Times, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[13] D. Li and R. Li. Preserving Operation Effects Relation in Group Editors. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2004*, pages 457–466, Chicago, Illinois, USA, November 2004. ACM Press.

[14] D. Li and R. Li. An Approach to Ensuring Consistency in Peer-to-Peer Real-Time Group Editors. *Computer-Supported Cooperative Work - JCSCW*, (to appear):1–59, 2006.

[15] R. Li and D. Li. A Landmark-Based Transformation Approach to Concurrency Control in Group Editors. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2005*, pages 284–293, Sanibel Island, Florida, USA, November 2005. ACM Press.

[16] R. Li and D. Li. Commutativity-Based Concurrency Control in Groupware. In *Proceedings of the IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2005*, pages 1–10, San Jose, California, USA, December 2005. IEEE Computer Society.

[17] B. Lushman and G. V. Cormack. Proof of Correctness of Ressel's adOPTed Algorithm. *Information Processing Letters*, 86(3):303–310, June 2003.

[18] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003*, pages 212–220, Sanibel Island, Florida, USA, November 2003. ACM Press.

[19] G. Oster, P. Urso, P. Molli, and A. Imine. Proving Correctness of Transformation Functions in Collaborative Editing Systems. Research Report RR-5795, LORIA – INRIA Lorraine, December 2005.

[20] G. Oster, P. Urso, P. Molli, and A. Imine. Data Consistency for P2P Collaborative Editing. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, Banff, Alberta, Canada, November 2006. ACM Press.

[21] A. Prakash and M. J. Knister. A Framework for Undoing Actions in Collaborative Systems. *ACM Transactions on Computer-Human Interaction*, 1(4):295–330, December 1994.

[22] M. Ressel and R. Gunzenhäuser. Reducing the Problems of Group Undo. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP'99*, pages 131–139, Phoenix, Arizona, USA, November 1999. ACM Press.

[23] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW'96*, pages 288–297, Boston, Massachusetts, USA, November 1996. ACM Press.

[24] Y. Saito and M. Shapiro. Optimistic Replication. *ACM Computing Surveys*, 37(1):42–81, 2005.

[25] SO6, an operational transformation-based synchronizer, (2006). Online http://www.libresource.org/.

[26] M. Suleiman, M. Cart, and J. Ferrié. Concurrent Operations in a Distributed and Mobile Collaborative Environment. In *Proceedings of the International Conference on Data Engineering - ICDE'98*, pages 36–45, Orlando, Florida, USA, February 1998. IEEE Computer Society.

[27] C. Sun. Undo as Concurrent Inverse in Group Editors. *ACM Transactions on Computer-Human Interaction*, 9(4):309–361, December 2002.

[28] C. Sun and C. Ellis. Operational Transformation in Real-Time Group Editors: Issues, Algorithms and Achievements. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW'98*, pages 59–68, Seattle, Washington, USA, November 1998. ACM Press.

[29] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, March 1998.

[30] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies Convergence in a Distributed Real-Time Collaborative Environment. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2000*, pages 171–180, Philadelphia, Pennsylvania, USA, December 2000. ACM Press.

[31] S. Weiss. Annulation de Groupe dans les Éditeurs Collaboratifs. Master's Thesis, Université Henri Poincaré - Nancy I, June 2006.